

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de ciencias y sistemas
Arquitectura de computadores y ensambladores 2
Sección D
Primer semestre 2025



Proyecto 1

Estudiante

Diego Felipe Cali Morales
Bruce Carbonell Castillo Cifuentes
Natalia Mariel Calderón Echeverría
Daniel Eduardo Velasquez Avila

Carnet

202201128
202203069
2022200007
202200041

Índice

Prototipo - Módulos	3
Processing	9
Maqueta	10
Props	11
Conexiones - Proceso	13
Stack IoT Framework	14
1. Capa de Percepción (Sensors)	14
2. Capa de Red (Connectivity)	14
3. Capa de Procesamiento (Analytics)	14
4. Capa de Aplicación (Smart Apps)	14
5. Capa de Infraestructura de Producto (Product Infrastructure)	15
Beneficios del Sistema	16
Impacto Ambiental	16
Link del repositorio	17

Prototipo - Módulos

- **Sensor ultrasónico**

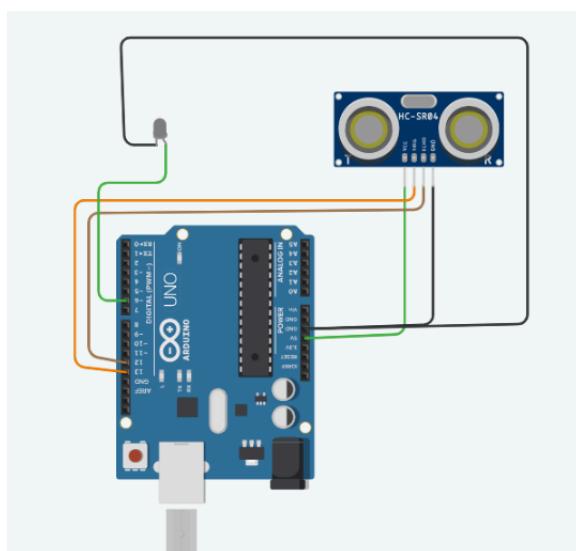
El uso del sensor ultrasónico se debe a la necesidad de detectar la presencia de una persona, esto mediante la medición de la distancia a la que se encuentra del propio sensor. Primero se asignaron los pines específicos para el LED y para el sensor ultrasónico, donde TRIG envía la señal de disparo y ECHO recibe el eco que indica la distancia.

```
int TRIG = 13;  
  
int ECHO = 12;
```

Posteriormente, en el área de setup se establecen los modos de entrada y salida para cada pin y se inicia la comunicación serial para la visualización de datos.

```
void setup(){  
  
    pinMode(TRIG, OUTPUT); //Salida  
    pinMode(ECHO, INPUT); //Entrada  
  
    pinMode(LED, OUTPUT);  
    Serial.begin(9600);  
}
```

En el loop, se activa el pin de Trigger para enviar un pulso ultrasónico muy breve. Luego, se mide el tiempo que tarda en recibir el eco mediante pulseIn en el pin ECHO. La distancia se calcula dividiendo el tiempo de eco entre 58.2 (para convertirlo a centímetros).



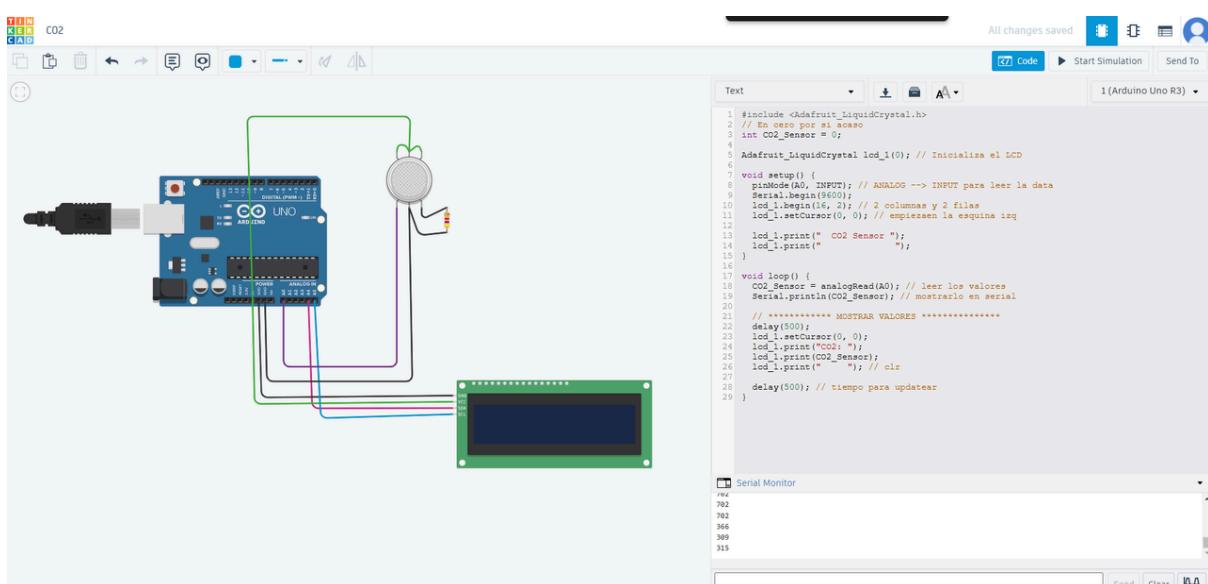
- **Sensor CO2**

En este caso se diseñó un circuito pude manejar un display LCD y mostrar la lectura de dicho sensor. Primero se establece la librería necesaria para controlar la LCD.

```
#include <LiquidCrystal_I2C.h>
```

Luego es necesario declarar una variable, en donde se almacenará la lectura analógica proveniente del pin analógico al que se encuentra conectado. En el setup, es necesario declarar el pin analógico como un pin de entrada para posteriormente poder leer los valores en el Loop.

```
CO2_Sensor = analogRead(A0);
```



- **Monitoreo Lumínico**

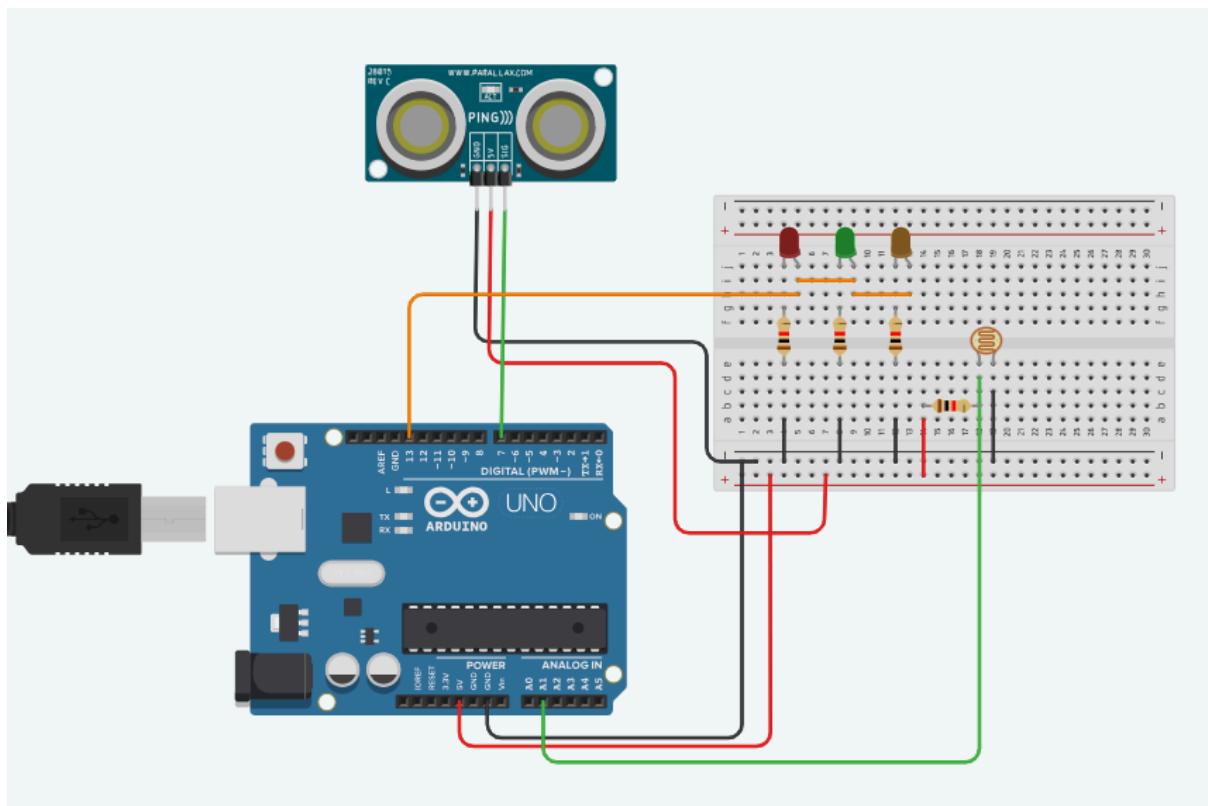
En este caso se buscaba añadir el monitoreo lumínico al funcionamiento, para esto al inicio se asignan los pines y variables necesarias.

```
const int pinLDR = A1;
luz (LDR) int valorLDR = 0;
```

Posteriormente, se define en la función functionController la lectura de dicho sensor y cuando ya se haya leído, el valor se imprime en el monitor serial junto con la medición de distancia:

```
valorLDR = analogRead(pinLDR);
Serial.print("Luz: ");
Serial.println(valorLDR);
```

Por ultimo, valor del sensor de luz se utiliza para determinar si el ambiente tiene "poca luz", si la intensidad de luz es baja ($\text{valorLDR} \leq 500$) y se detecta un objeto a 200 cm o menos, se activan los LED



- **Sensor de temperatura y humedad**

Para el monitoreo de temperatura y humedad al inicio se define el pin y el tipo del sensor DHT22, y se crea un objeto dht para gestionar las lecturas.

```
// Configuración del sensor DHT22
#define DHTPIN A3
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
```

Luego se obtienen los valores de dichas lecturas y se establece el manejo de errores en caso de que alguna lectura no sea válida.

```

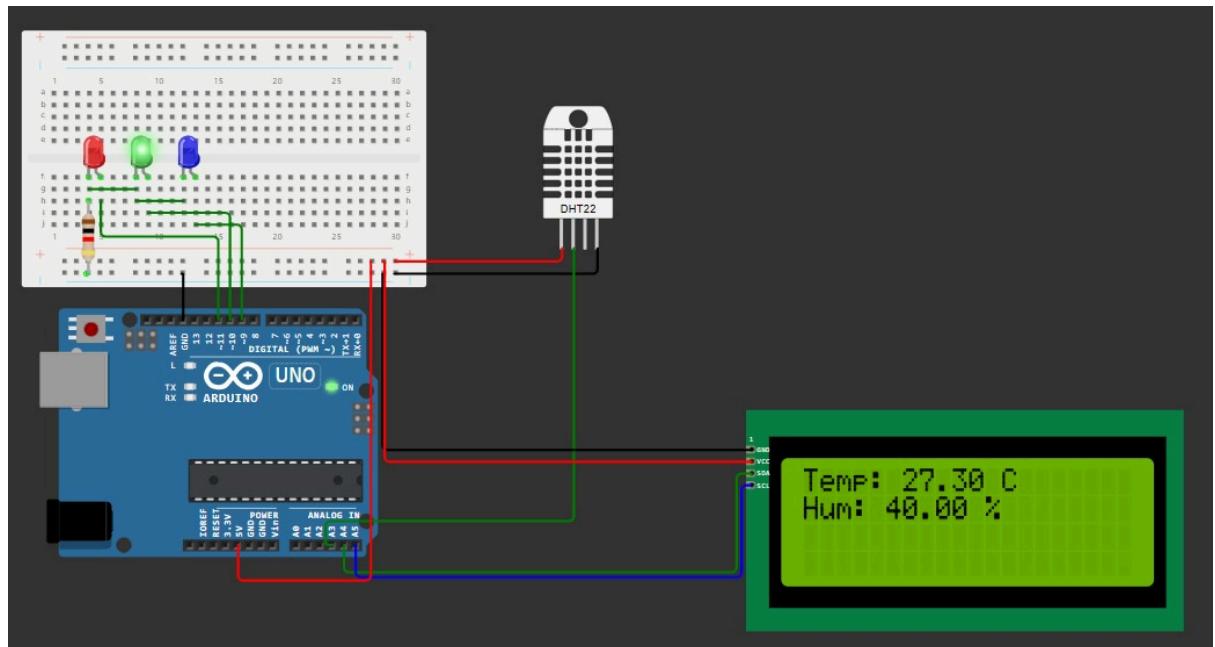
float humedad = dht.readHumidity();
float temperatura = dht.readTemperature();

// Verificar si la lectura es válida
if (isnan(temperatura) || isnan(humedad)) {
    lcd.setCursor(0, 0);
    lcd.print("Error leyendo");
    lcd.setCursor(0, 1);
    lcd.print("DHT22...");
    delay(2000);
    return -1;
}

```

Una vez validados, se muestran en pantalla los valores actuales de temperatura y humedad. Según la temperatura medida se cada una de las distintas LEDs.

- $\geq 30^\circ\text{C}$: Se enciende el LED rojo para indicar una temperatura alta.
- $\leq 18^\circ\text{C}$: Se enciende el LED azul para indicar una temperatura baja.
- Entre 18°C y 30°C : Se enciende el LED verde para indicar una temperatura moderada.



- **Boceto Keypad**

El objetivo del diseño era crear el menú inicial necesario para la práctica a través del keypad y probar el mismo con algunos de los sensores ya trabajados. El menú inicial permite:

- **Navegar entre opciones:**

Permite alternar entre "DATA en vivo" y "Historial EEPROM".

- **Seleccionar opciones:**

Con la tecla de selección se muestra la opción elegida (por ejemplo, visualizar datos en vivo o recuperar datos almacenados).

- **Ejecutar acciones adicionales:**

Algunas teclas permiten guardar datos en la EEPROM o regresar al menú

Al inicio se define una matriz de 4 filas por 4 columnas, se asignan los caracteres correspondientes y se especifican los pines conectados.

```
const byte numRows = 4; // Filas
const byte numCols = 4; // Columnas

// Mapeo de teclas
char keymap[numRows][numCols] =
{
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

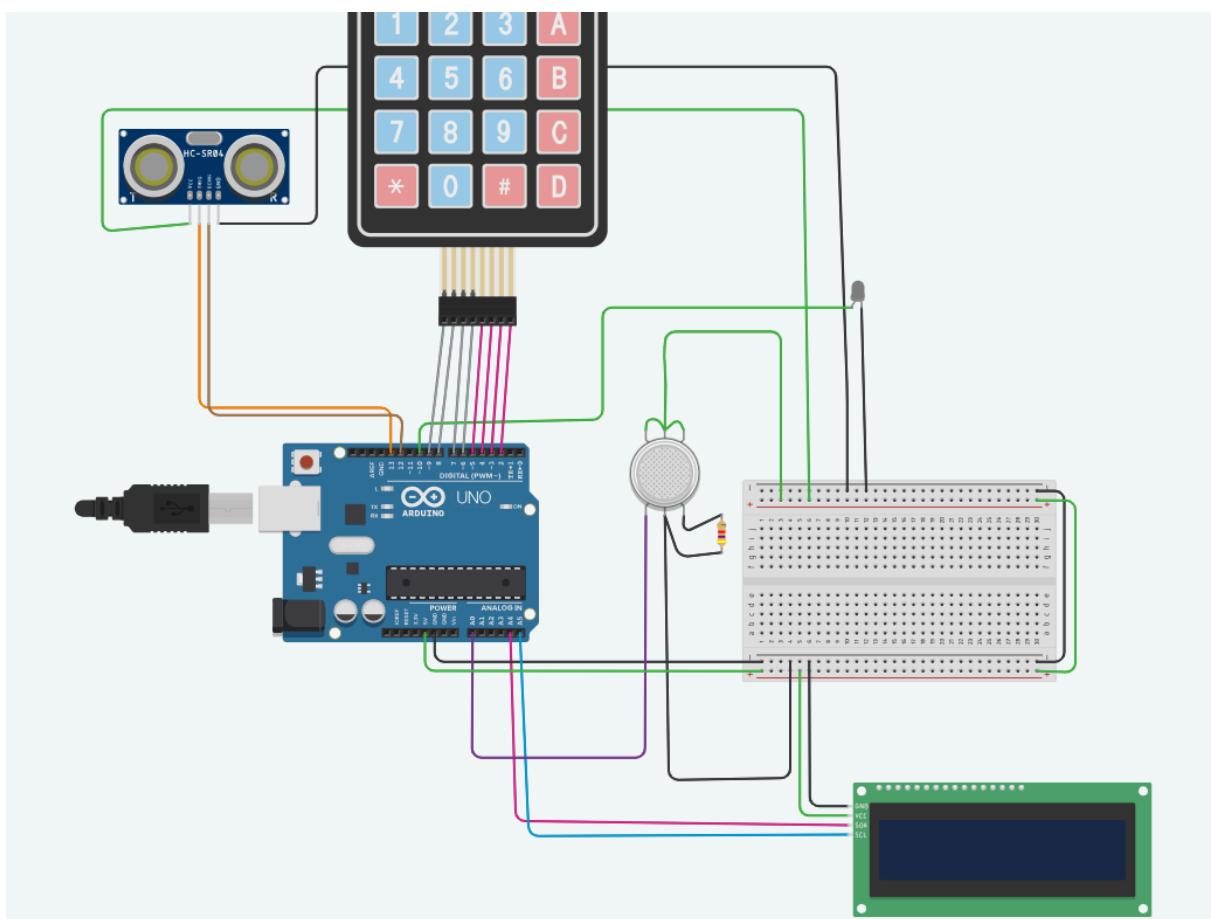
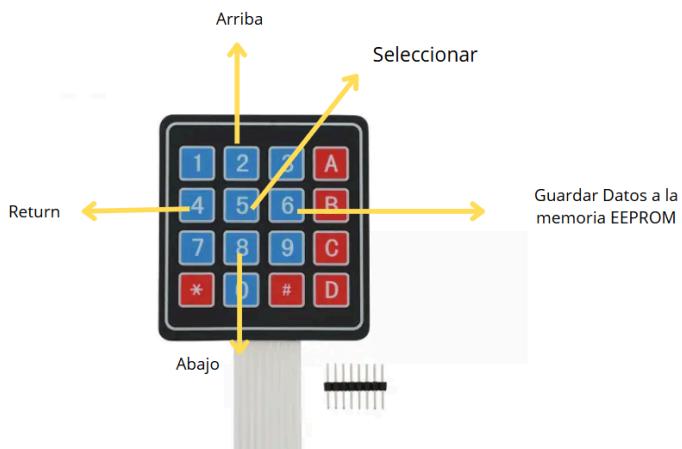
// Pines para filas y columnas
byte rowPins[numRows] = {9, 8, 7, 6};
byte colPins[numCols] = {5, 4, 3, 2};

// Instancia del Keypad
Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);
```

Posteriormente, se lee continuamente la tecla presionada y de acuerdo al valor de la misma se realizan actividades específicas.

```
lcd_key = myKeypad.getKey();
```

- Keypad Layout



Processing

Para esta práctica se hizo uso de un archivo de processing para crear una interfaz que muestre en tiempo real los datos enviados a través del puerto serial. El formato fue el siguiente:

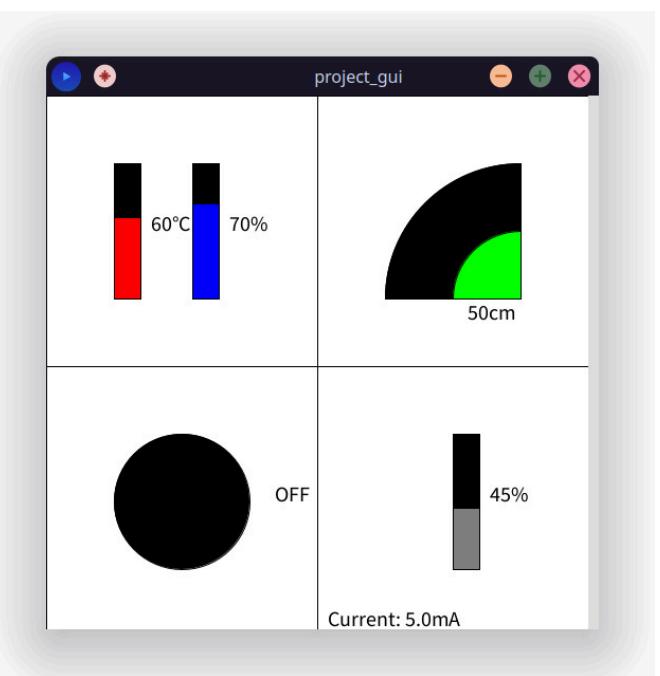
```
temperature;humidity;distance;light;co2\n
```

Al inicio, en el `Setup()` se configura una ventana de 512x 512 pixeles y se indica que se debe esperar un salto de línea '\n' para el procesamiento de cada conjunto de datos.

```
void setup() {
    size(512, 512);
    try {
        myPort = new Serial(this, "/dev/ttyACM0", 9600);
        myPort.bufferUntil('\n');
    } catch (Exception e){
        println("Error opening serial port");
    }
}
```

En cada una de las iteraciones de `draw()`, en caso se reciba una cadena valida, esta se separa en los cinco componentes especificados al inicio. Visualización La ventana se divide en 4 cuadrantes:

1. Cuadrante superior izquierdo Se encarga de mostrar la temperatura y el nivel de humedad detectado.
2. Cuadrante superior derecho En este cuadrante se muestra la distancia medida, para esto se usa la función "drawRadar".
3. Cuadrante inferior izquierdo Para mostrar el estado del sensor de luz se dibuja una bombilla que se ilumina dependiendo del estado del sensor de luz.
4. Cuadrante inferior derecho En este cuadrante se muestra el resultado de la medición de CO2 en porcentaje .

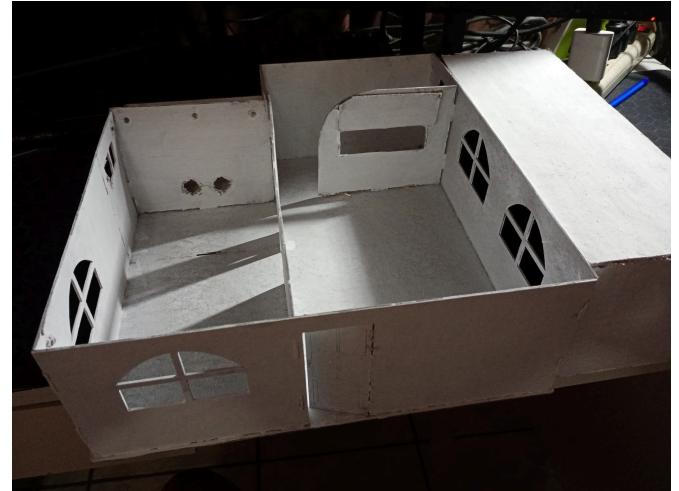
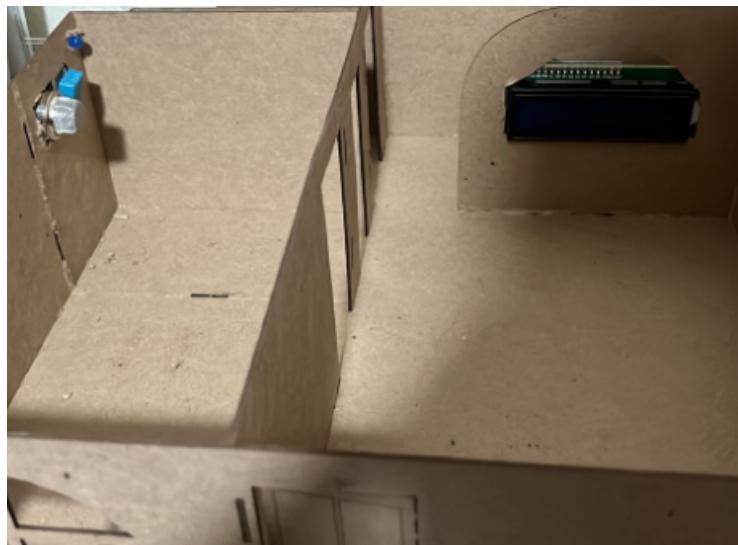
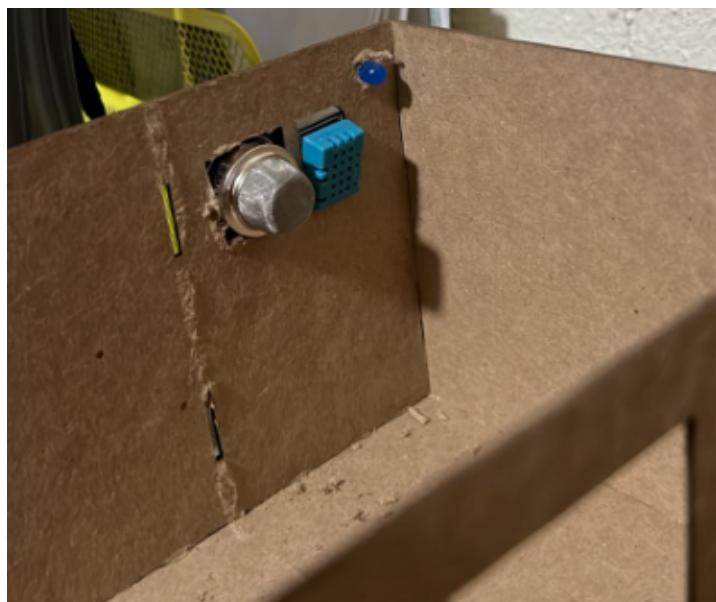


Maqueta

El objetivo principal de la maqueta consistía en desarrollar un Sistema Inteligente de Monitoreo Ambiental para Cuartos de Servidores mediante IoT. Por ello, se estableció que la maqueta debía ser lo suficientemente espaciosa para que la presencia de los sensores no redujera el espacio disponible ni afectaría la estética visual.

Tomando en cuenta los sensores a utilizar, se determinaron dimensiones de 29×27 centímetros. A partir de ahí, se inició el proceso de diseño de los ambientes y la distribución de los sensores.

Los sensores fueron colocados en lugares estratégicos para garantizar su correcto funcionamiento sin comprometer la armonía visual. Se buscó que su presencia pasara lo más desapercibida posible, manteniendo un equilibrio entre funcionalidad y estética. Así comenzó el proceso de instalación de los sensores:



Props

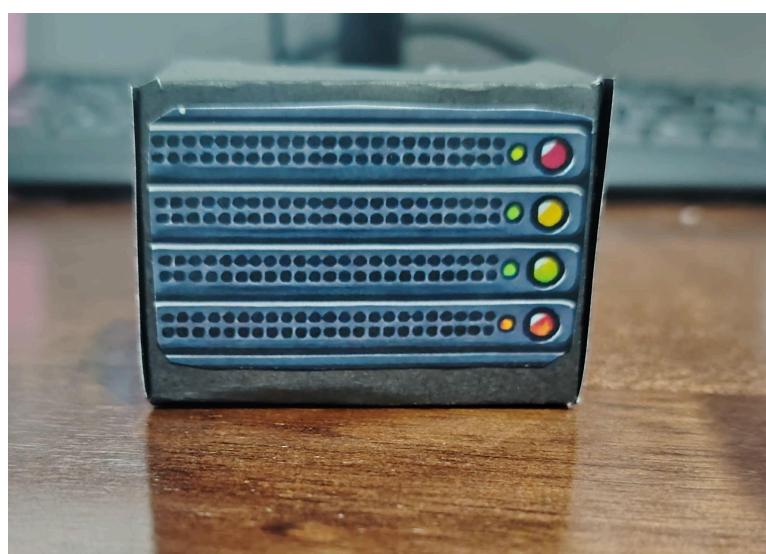
Debido a la naturaleza de la maqueta, es necesario el uso de material que ayude a simular de forma correcta el espacio que se quiere replicar.

Es por eso que se elaboraron torres de servidores, con el objetivo de ambientar y mejorar la presentación de la maqueta.

- **Torre 1**



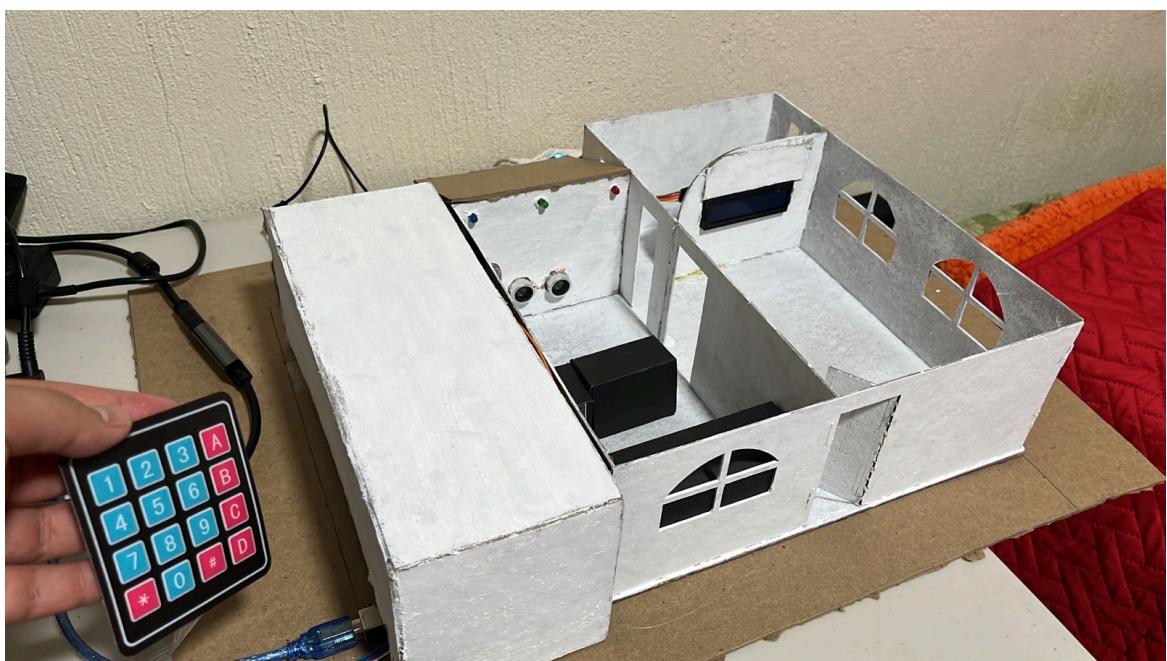
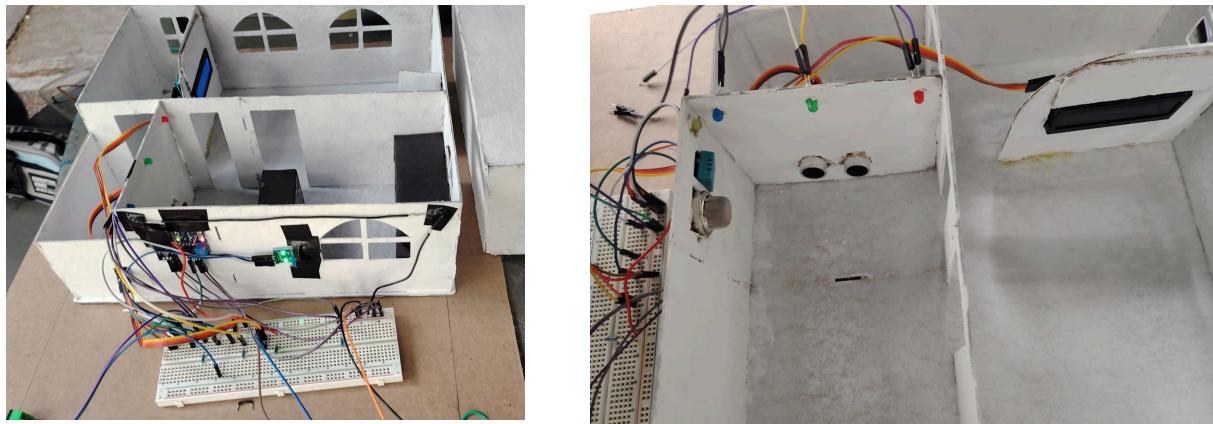
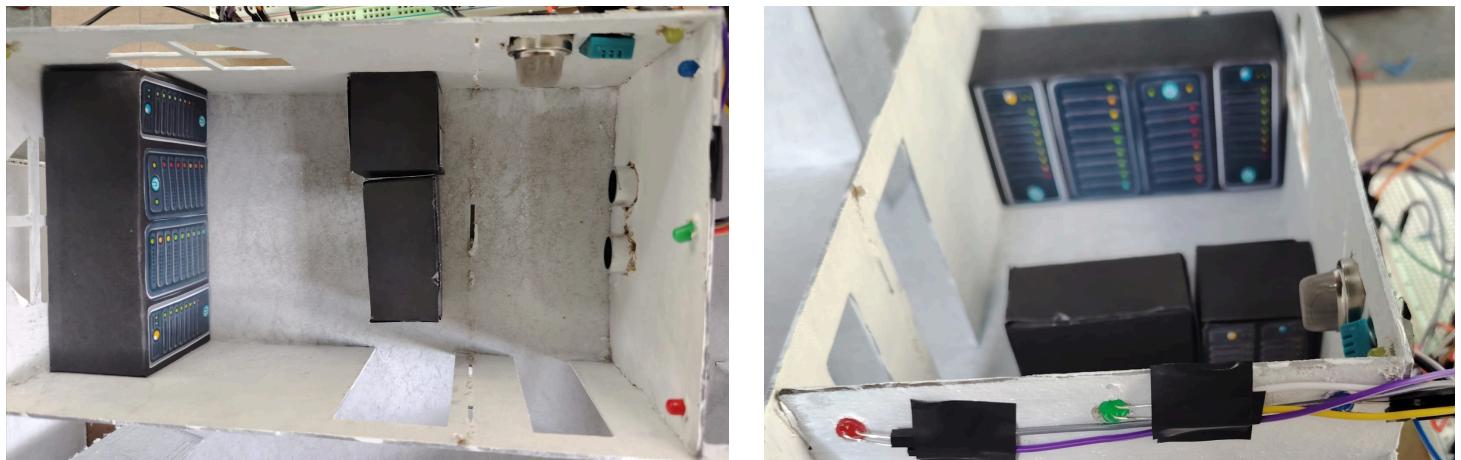
- **Torre 2**



- **Torre 3**



Conexiones - Proceso



Stack IoT Framework Stack IoT Framework

Para organizar la arquitectura del sistema, se utilizó el Stack IoT Framework, que se compone de varias capas que facilitan la interacción entre los sensores, el procesamiento de datos y la comunicación con la nube. A continuación, se describen las capas involucradas:

1. Capa de Percepción (Sensors)

Esta es la primera capa del sistema y se encarga de la adquisición de datos mediante los sensores. Aquí se incluyen:

- Sensor ultrasónico para detección de proximidad.
- Sensor de temperatura y humedad DHT22.
- Sensor de CO2.
- Sensor LDR para monitoreo lumínico.

Estos sensores capturan información relevante sobre el entorno y la transmiten al sistema para su análisis.

2. Capa de Red (Connectivity)

Esta capa se encarga de la transmisión de datos entre los sensores y el microcontrolador. En este proyecto, se utiliza la comunicación serie con Serial.begin(9600); para enviar los datos capturados al sistema central, permitiendo su procesamiento y visualización.

3. Capa de Procesamiento (Analytics)

En esta capa, el microcontrolador interpreta los datos recibidos y ejecuta las acciones necesarias. Entre sus funciones principales destacan:

- Análisis de la distancia detectada por el sensor ultrasónico.
- Determinación de la calidad del aire mediante el sensor de CO2.
- Control de los LEDs en función de los valores de temperatura y luminosidad.
- Almacenamiento temporal de los datos para su posterior análisis.

4. Capa de Aplicación (Smart Apps)

Esta última capa proporciona la interfaz con el usuario y permite visualizar los datos obtenidos en tiempo real. En este caso, se emplean dos medios principales:

- Display LCD: Muestra valores en tiempo real, como temperatura y humedad.
- Software Processing: Se utiliza para presentar gráficamente la información recolectada en una interfaz interactiva.

El Stack IoT Framework permite la modularidad y escalabilidad del sistema, asegurando un flujo de datos eficiente desde la captura hasta la presentación final. Gracias a esta estructura, el monitoreo ambiental se vuelve más preciso y adaptable a diferentes condiciones y entornos.

5. Capa de Infraestructura de Producto (Product Infrastructure)

Esta capa abarca la base tecnológica y física necesaria para soportar el sistema IoT. También, como su nombre lo indica, la infraestructura del mismo, que en este caso se vería reflejado en la maqueta elaborada. Usualmente en esta capa se incluyen:

- **Servidores**: Para gestionar y almacenar grandes volúmenes de datos.
- **Centros de datos y edge computing**: Para procesamiento local y reducción de latencia.
- **Plataformas IoT escalables**: Para administrar múltiples dispositivos y sensores en tiempo real.
- **Protocolos de seguridad y cifrado**: Para garantizar la protección de la información y el acceso seguro.

Beneficios del Sistema

- Monitoreo en Tiempo Real: Permite la supervisión continua de las condiciones ambientales en un cuarto de servidores.
- Automatización de Respuestas: Activa alarmas o luces de advertencia en función de las mediciones.
- Optimización del Uso de Energía: La activación de dispositivos solo ocurre cuando es necesario.
- Reducción de Riesgos: Minimiza fallos en equipos al controlar la temperatura y la humedad.
- Accesibilidad Remota: Puede integrarse con sistemas en la nube para acceder a la información desde cualquier lugar.
- Escalabilidad: Se pueden agregar más sensores para ampliar la funcionalidad.

Impacto Ambiental

- Eficiencia Energética: Reduce el consumo de energía al operar bajo demanda.
- Menos Emisión de Calor: Ayuda a controlar la temperatura y evitar sobrecalentamiento.
- Optimización de Recursos: Reduce la necesidad de intervención manual para monitoreo.
- Mayor Durabilidad de Equipos: Mantener condiciones óptimas extiende la vida útil de los servidores.

Diagrama Funcionamiento

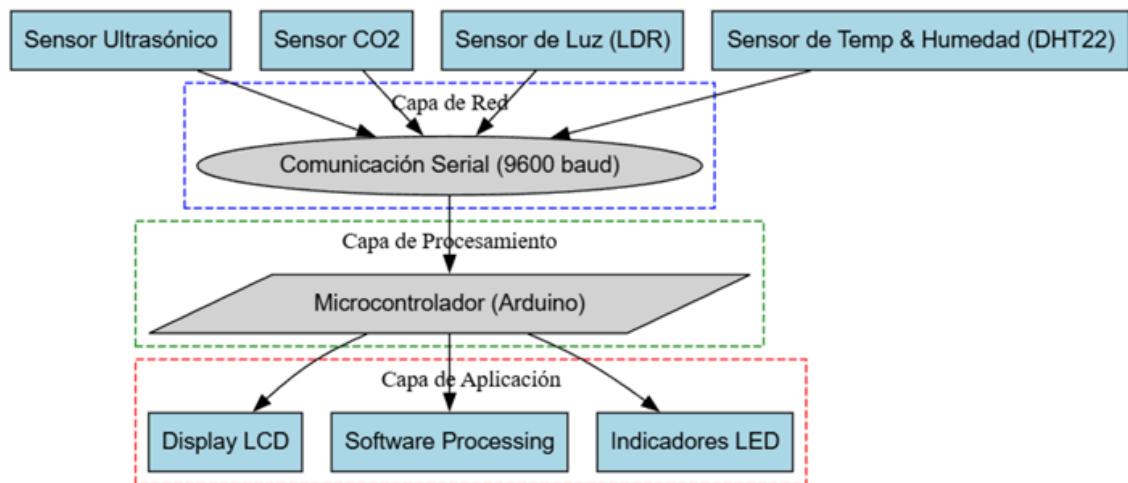
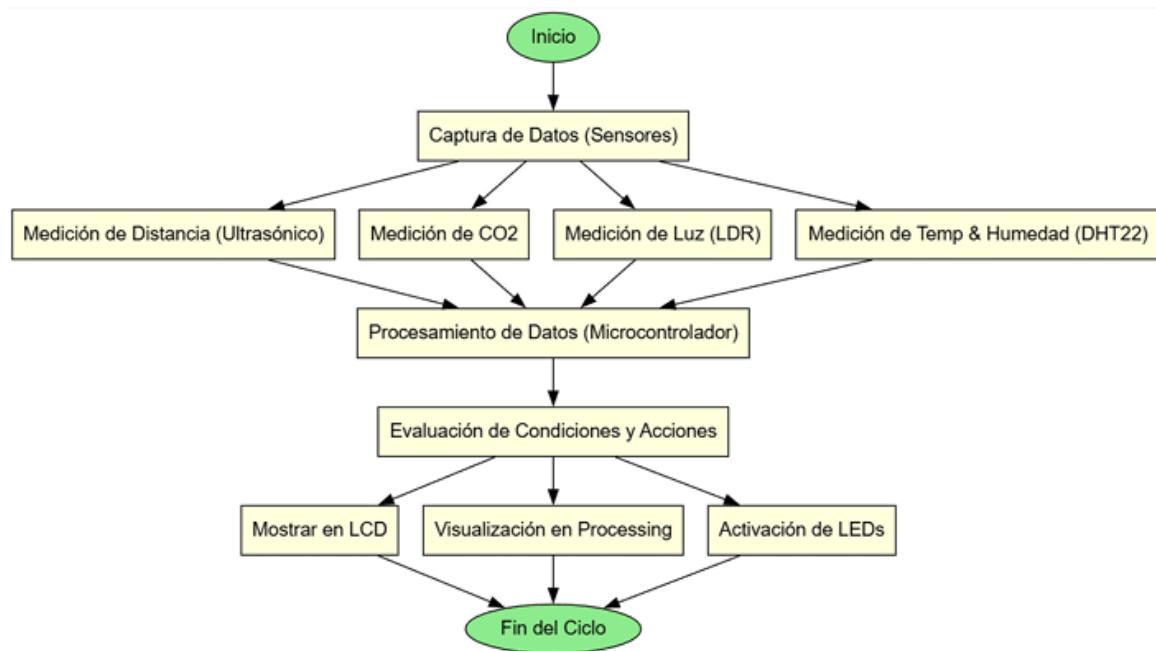


Diagrama Procesos



Link del repositorio

https://github.com/DiegoCali/ARQUI2B_1S2025_G9.git